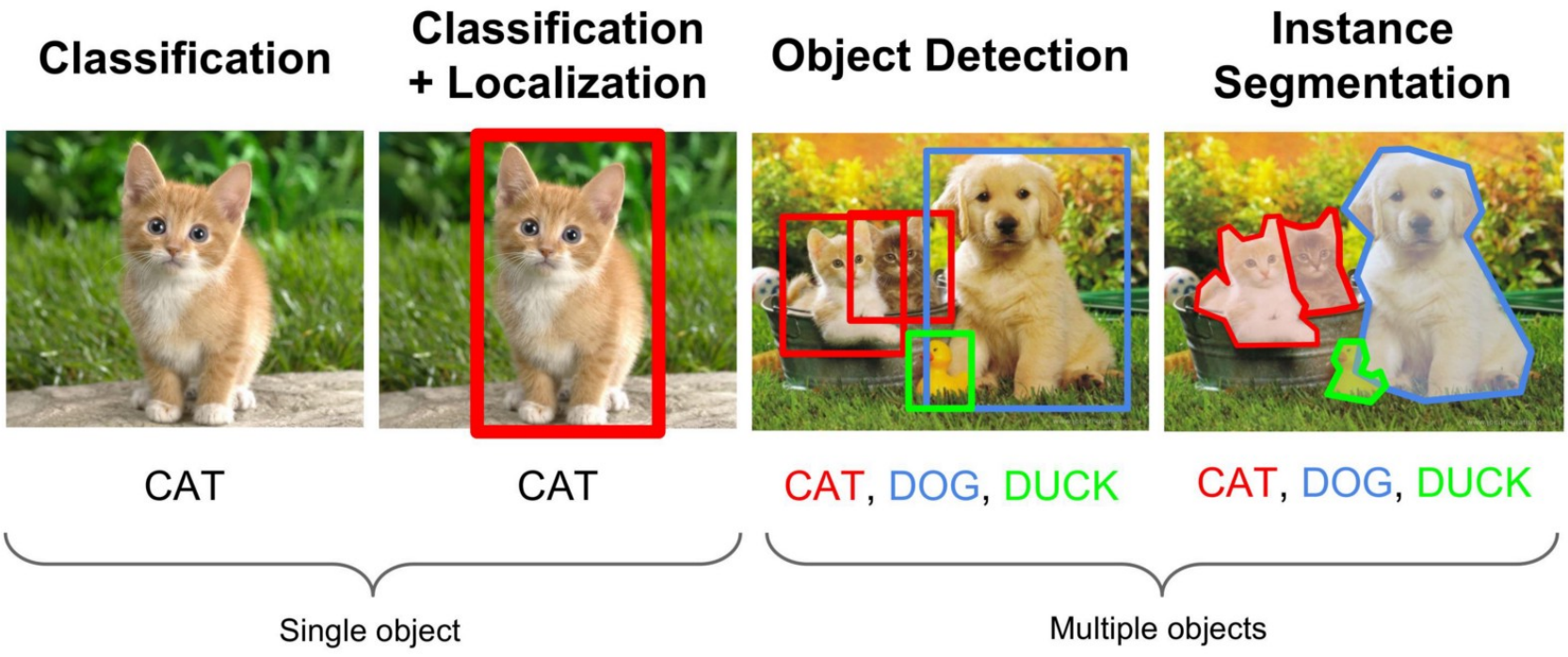
Background on Deep Learning

Nate Merrill

University of Delaware

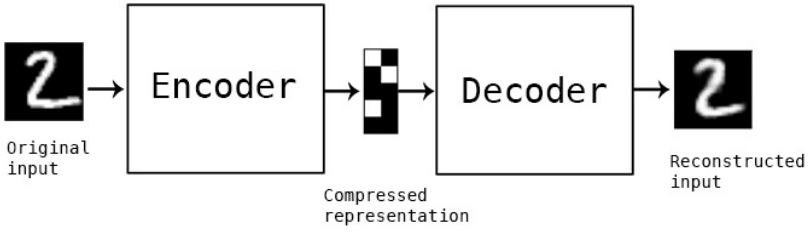
Popular Deep Learning Tasks

- Classification
- Classifcation + Bounding Box Detection
- Muti-object detection
- Pixel-wise semantic segmentation

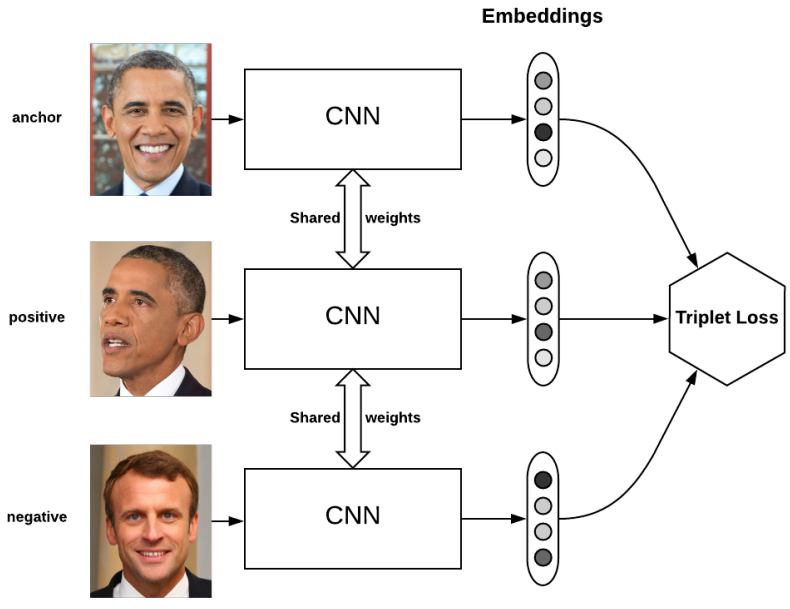


More Popular Deep Learning Tasks

- Embeddings (dimension reduction)
  - Autoencoder
  - Triplet Embedding



Autoencoder



Triplet embedding

Fully Connected Layer

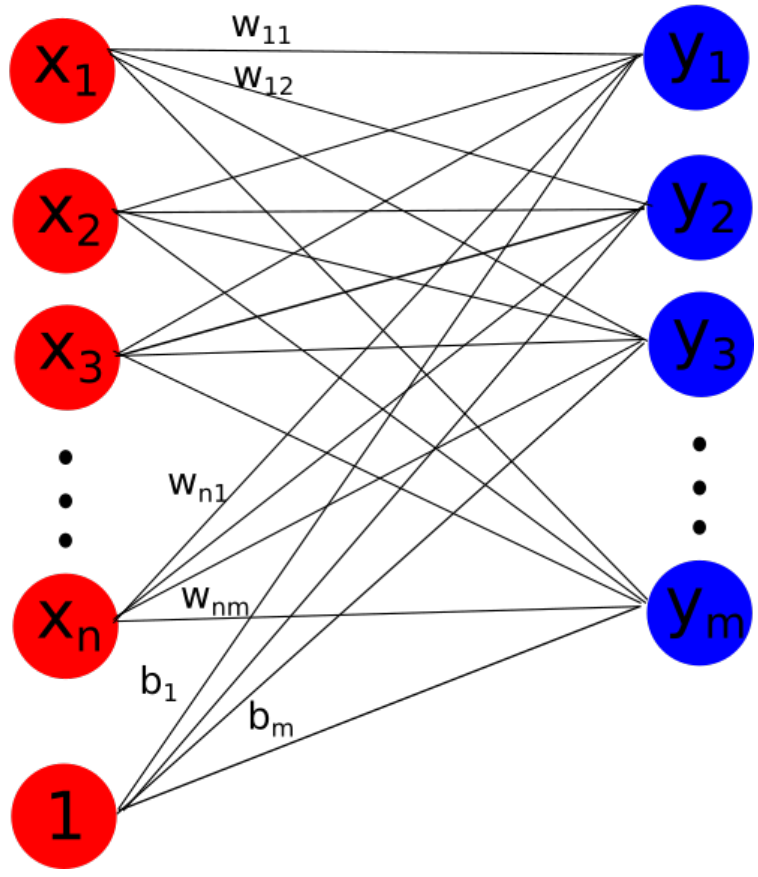
- Vector input and output
- Learned weights associated with each connection
- Can be written as a linear operation

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n & 1 \end{bmatrix}^\mathsf{T}$$

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix}^\mathsf{T}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} & b_1 \\ w_{21} & w_{22} & \dots & w_{2n} & b_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} & b_m \end{bmatrix}$$

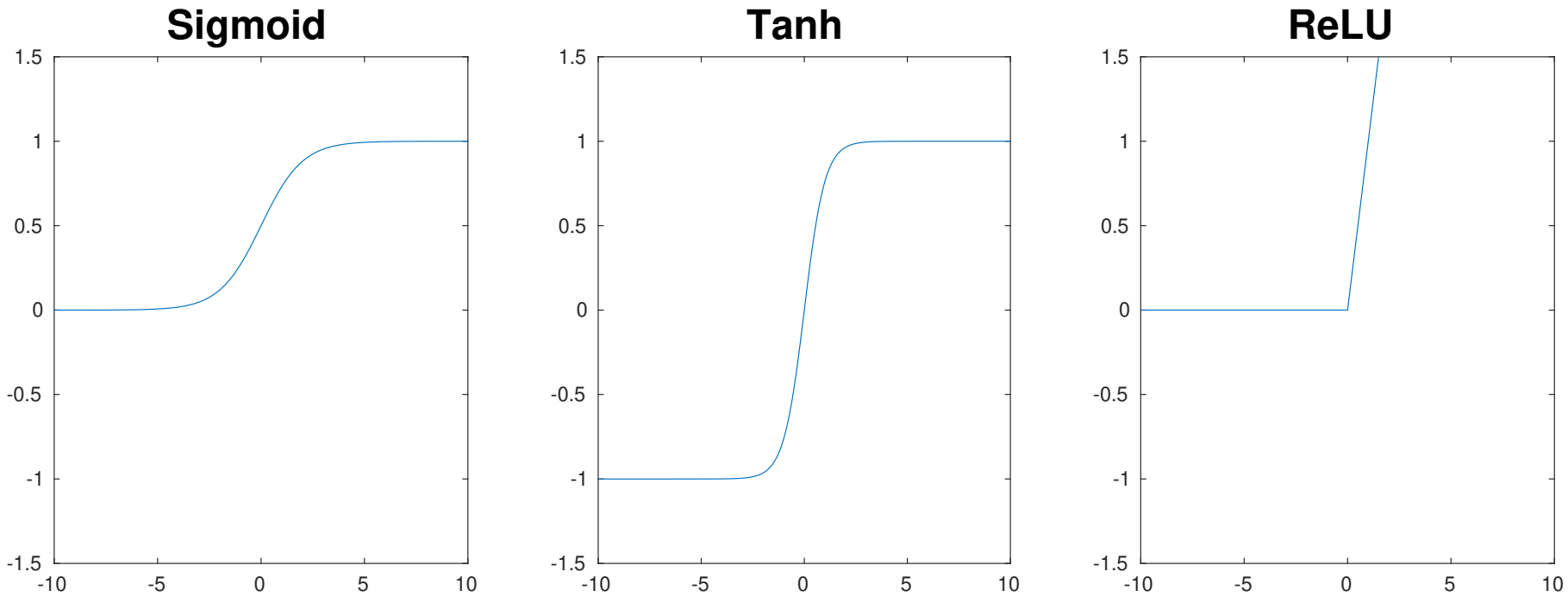
$$\mathbf{y} = \mathbf{W}\mathbf{x}$$



Nonlinear Activation

- Fully connected layer cannot model nonlinear functions
- Nonlinear activations are used to provide nonlinearity
- Key idea: they resemble a "neuron" firing

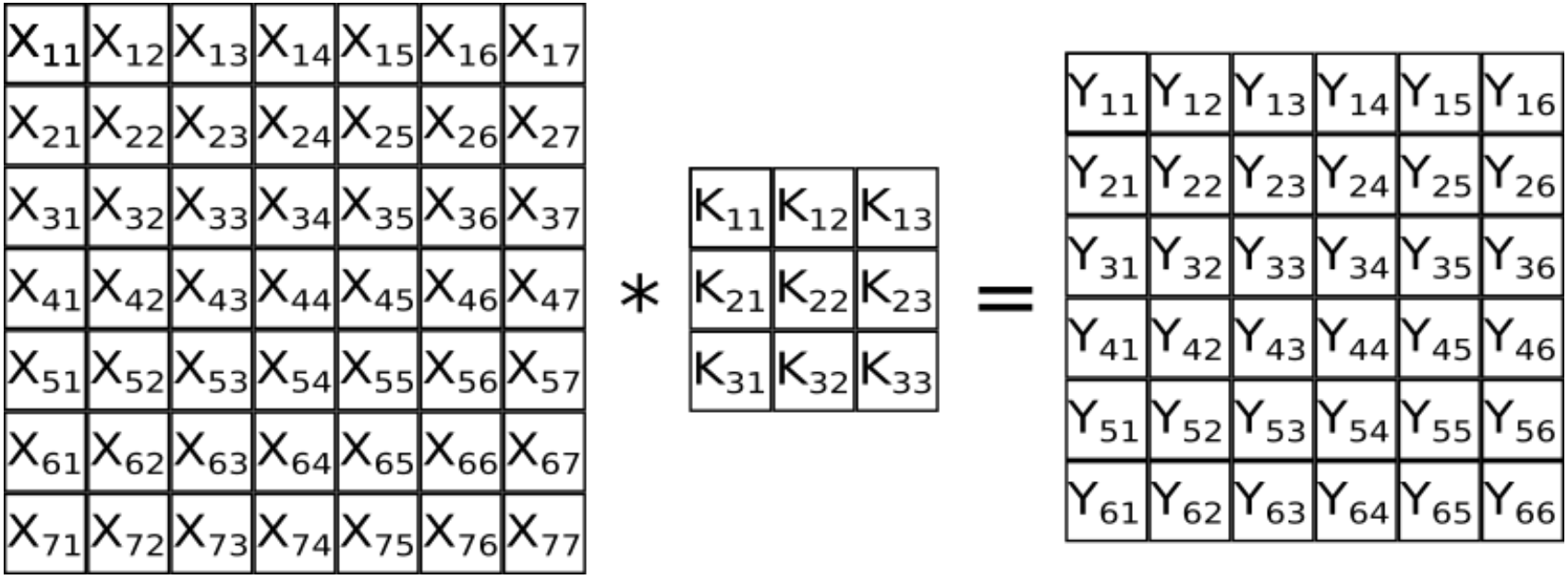
Examples:



Popular activation functions

Convolution Layer

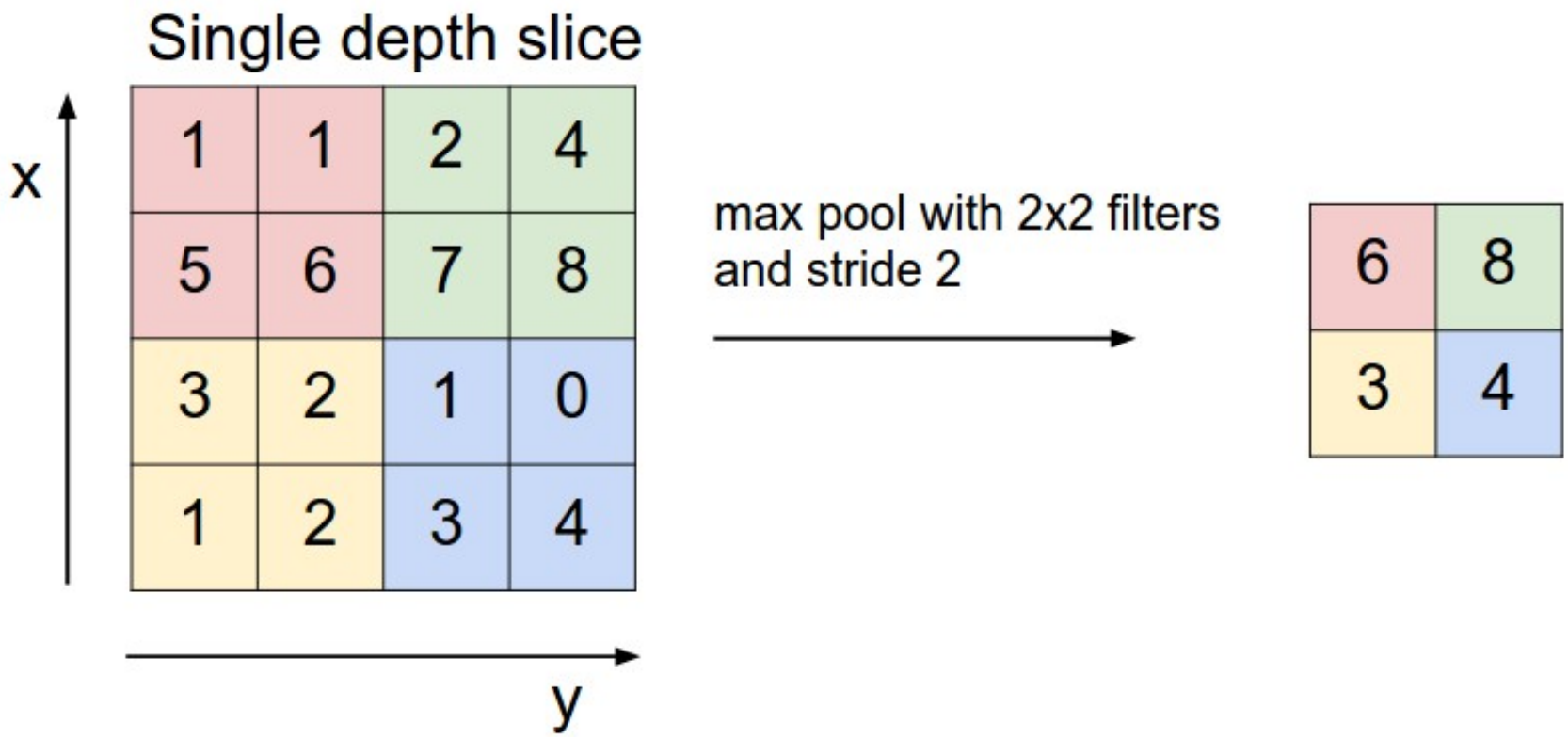
- Matrix/Tensor input and output
- Useful for image input



$$
\begin{aligned}
Y_{ij} = {} & K_{33}X_{i,j} + K_{32}X_{i,j+1} + K_{31}X_{i,j+2} + K_{23}X_{i+1,j} \\
& + K_{22}X_{i+1,j+1} + K_{21}X_{i+1,j+2} + K_{13}X_{i+2,j} \\
& + K_{12}X_{i+2,j+1} + K_{11}X_{i+2,j+2} + b
\end{aligned}
$$

Max Pooling Layer

- Useful for translational invariance
- Similar operation to convolution



Softmax Layer

- Bounds output to [0,1]
- Sum of output is 1
- Useful for learning probability mass functions

Suppose $\mathbf{x} \in \mathbb{R}^n$, then

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Loss Function

- A scalar error metric for the network
- Necessary for training. Otherwise there is no optimization criteria

Popular examples:

Cross Entropy
Suppose $\mathbf{x} \in \{0,1\}^n$ is the ground truth class labels and $\hat{\mathbf{x}} \in [0,1]^n$ s.t. $\sum_{i=1}^{n} \hat{x}_i = 1$ is the output of a *classification* network, then

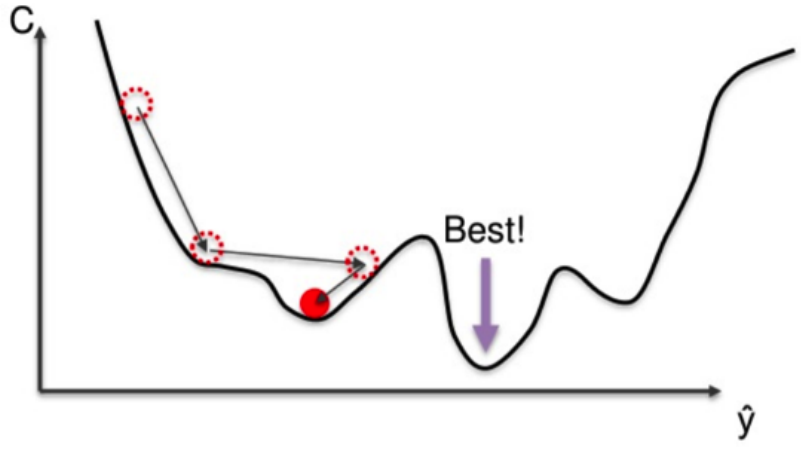
$$E(\mathbf{x}, \hat{\mathbf{x}}) = -\sum_{i=1}^{n} x_i \ln(\hat{x}_i)$$

MSE Suppose $\mathbf{x} \in \mathbb{R}^n$ is the ground truth and $\hat{\mathbf{x}} \in \mathbb{R}^n$ is the output of a *regression* network, then

$$E(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{n} ||\hat{\mathbf{x}} - \mathbf{x}||_2^2$$

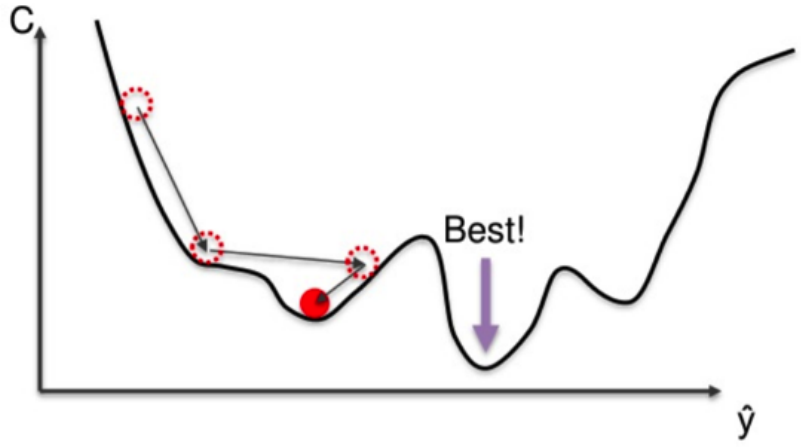
Optimization

- Suppose we have a feed-forward neural network with arbitrary tensor input $\boldsymbol{X}$, output (loss/error) E, and layers $L_1, L_2, \ldots, L_k$ (including loss layer).
- How can we optimize such a complicated function?
- There are many local minima to get caught in



Optimization

- Suppose we have a feed-forward neural network with arbitrary tensor input $\boldsymbol{X}$, output (loss/error) E, and layers $L_1, L_2, \ldots, L_k$ (including loss layer).
- How can we optimize such a complicated function?
- There are many local minima to get caught in
  - **Idea:** Look at each layer separately
  - Observe that $E = L_k(L_{k-1}(\ldots(L_1(\boldsymbol{X}))))$.
  - We can estimate the gradient of the entire network with the chain rule!



Optimization: Stochastic Gradient Descent

- Create copies of your network
  - Train on multiple data samples at once (i.e., a *batch*)
  - Average out the loss values across examples
  - Make sure the examples are selected *randomly*
- At each iteration, estimate the gradient of each layer w.r.t. the weights
- Update the weights with the *average* gradient over the batch with a scalar multiplier called the *learning rate* (typically $<< 1$)

Suppose we have the weights of layer L_i, at iteration j, $\boldsymbol{W}_i^{(j)}$, with layer input $\boldsymbol{X}$, learning rate η, and n examples per batch. Then the update step is:

$$\boldsymbol{W}_i^{(j+1)} = \boldsymbol{W}_i^{(j)} - \frac{\eta}{n} \sum_{k=1}^{n} \frac{\partial L_i(\boldsymbol{W}_i^{(j)}, \boldsymbol{X}_k)}{\partial \boldsymbol{W}_i^{(j)}}$$

Implementation

Popular Libraries:

- **TensorFlow**
- Keras
- Caffe (Caffe2 as well)
- PyTorch
- MXNet
- Darknet

The slides and code for the tutorial can be found at:

`https://github.com/nmerrill67/DeepLearningTutorial`

We will step through the script together.